COMS W4705, Spring 2015: Problem Set 2

Total points: 140

Analytic Problems (due March 2nd)

Question 1 (20 points)

A probabilistic context-free grammar $G = (N, \Sigma, R, S, q)$ in Chomsky Normal Form is defined as follows:

- N is a set of non-terminal symbols (e.g., `NP`, `VP`, `S` etc.)
- Σ is a set of terminal symbols (e.g., *cat, dog, the*, etc.)
- R is a set of rules which take one of two forms:
  - $X \rightarrow Y_1 Y_2$ for $X \in N$, and $Y_1, Y_2 \in N$
  - $X \rightarrow Y$ for $X \in N$, and $Y \in \Sigma$
- $S \in N$ is a distinguished start symbol
- q is a function that maps every rule in R to a probability, which satisfies the following conditions:
  - $\forall r \in R,\ q(r) \geq 0$
  - $\forall X \in N,\ \sum_{X \rightarrow \alpha \in R} q(X \rightarrow \alpha) = 1$

Now assume we have a probabilistic CFG G', which has a set of rules R which take one of the two following forms:

- $X \rightarrow Y_1 Y_2 \ldots Y_n$ for $X \in N$, $n \geq 2$, and $\forall i,\ Y_i \in N$
- $X \rightarrow Y$ for $X \in N$, and $Y \in \Sigma$

Note that this is a more permissive definition than Chomsky normal form, as some rules in the grammar may have more than 2 non-terminals on the right-hand side. An example of a grammar that satisfies this more permissive definition is as follows:

S	→	NP	VP		0.7
S	→	NP	NP	VP	0.3
VP	→	Vt	NP		0.8
VP	→	Vt	NP	PP	0.2
NP	→	DT	NN	NN	0.3
NP	→	NP	PP		0.7
PP	→	IN	NP		1.0

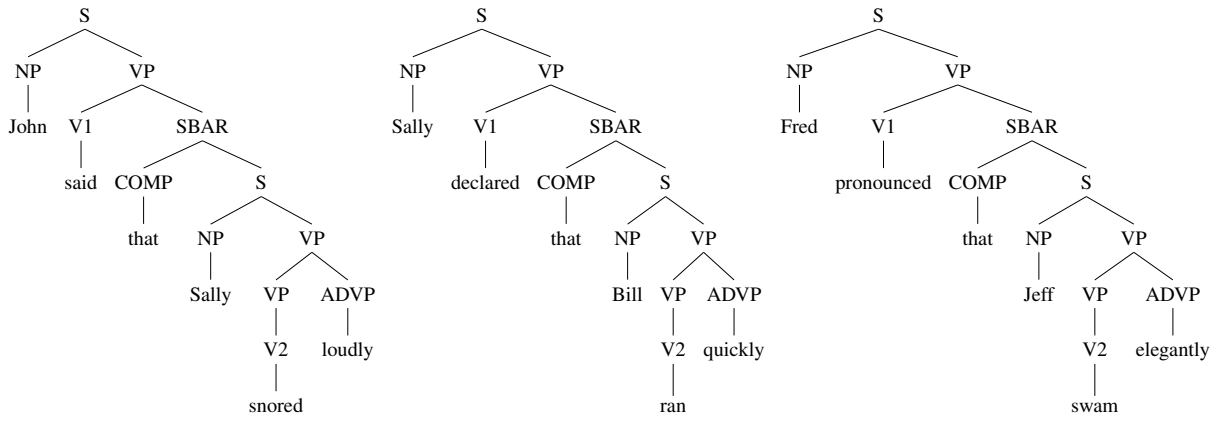
Vt	→	saw	1.0
NN	→	man	0.7
NN	→	woman	0.2
NN	→	telescope	0.1
DT	→	the	1.0
IN	→	with	0.5
IN	→	in	0.5

Question 1(a): Describe how to transform a PCFG G', in this more permissive form, into an "equivalent" PCFG G in Chomsky normal form. By equivalent, we mean that there is a one-to-one function f between derivations in G' and derivations in G, such that for any derivation T' under G' which has probability p, $f(T')$ also has probability p. (Note: one major motivation for this transformation is that we can then apply the dynamic programming parsing algorithm, described in lecture, to the transformed grammar.) Hint: think about adding new rules with new non-terminals to the grammar.

Question 1(b): Show the resulting grammar G after applying your transformation to the example PCFG shown above.

Question 2 (20 points)

Nathan L. Pedant decides to build a treebank. He finally produces a corpus which contains the following three parse trees:



Clarissa Lexica then purchases the treebank, and decides to build a PCFG, and a parser, using Nathan's data.

Question 2(a): Show the PCFG that Clarissa would derive from this treebank.

Question 2(b): Show two parse trees for the string "Jeff pronounced that Fred snored loudly", and calculate their probabilities under the PCFG.

Question 2(c): Clarissa is shocked and dismayed, (see 2(b)), that "Jeff pronounced that Fred snored loudly" has two possible parses, and that one of them—that Jeff is doing the pronouncing loudly—has relatively high probability, in spite of it having the `ADVP` *loudly* modifiying the "higher" verb, *pronounced*. This type of high attachment is never seen in the corpus, so the PCFG is clearly missing something. Clarissa decides to fix the treebank, by altering some non-terminal labels in the corpus. Show one such transformation that results in a PCFG that gives zero probability to parse trees with "high" attachments. (Your solution should systematically refine some non-terminals in the treebank, in a way that slightly increases the number of non-terminals in the grammar, but allows the grammar to capture the distinction between high and low attachment to VPs.)

Question 3 (20 points)

Consider the CKY algorithm for finding the maximum probability for any tree when given as input a sequence of words $x_1, x_2, \ldots, x_n$. As usual, we use N to denote the set of non-terminals in the grammar, and S to denote the start symbol.

The base case in the recursive definition is as follows: for all $i = 1 \ldots n$, for all $X \in N$,

$$\pi(i, i, X) = \begin{cases} q(X \rightarrow x_i) & \text{if } X \rightarrow x_i \in R \\ 0 & \text{otherwise} \end{cases}$$

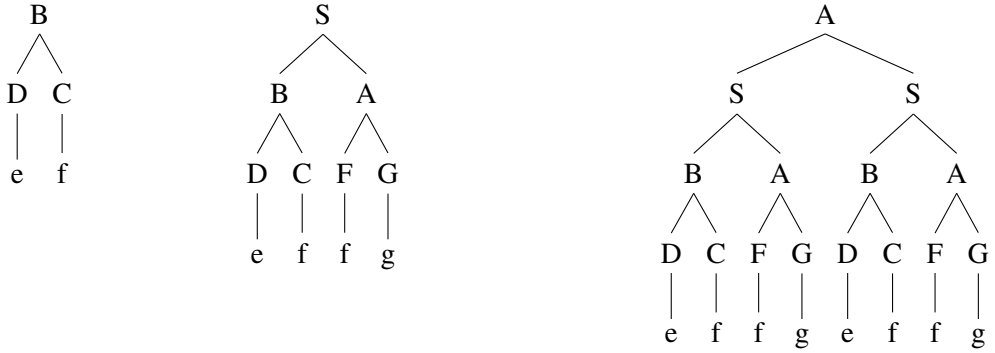
and the recursive definition is as follows: for all (i, j) such that $1 \leq i < j \leq n$, for all $X \in N$,

$$\pi(i, j, X) = \max_{\substack{X \rightarrow YZ \in R, \\ s \in \{i \ldots (j-1)\}}} (q(X \rightarrow YZ) \times \pi(i, s, Y) \times \pi(s+1, j, Z))$$

Finally, we return

$$\pi(1, n, S) = \max_{t \in \mathcal{T}_G(s)} p(t)$$

Now assume that we want to find the maximum probability for any *balanced* tree for a sentence. Here are some example balanced trees:



It can be seen that in balanced trees, whenever a rule of the form `X -> Y Z` is seen in the tree, then the non-terminals `Y` and `Z` dominate the same number of words.

NOTE: we will assume that the length of the sentence, n, is $n = 2^x$ for some integer x. For example $n = 2$, or 4, or 8, or 16, etc.

Question 3(a): Complete the recursive definition below, so that the algorithm returns the maximum probability for any **balanced** tree underlying a sentence $x_1, x_2, \ldots, x_n$.

Base case: for all $i = 1 \ldots n$, for all $X \in N$,

$$\pi(i, i, X) = \begin{cases} q(X \rightarrow x_i) & \text{if } X \rightarrow x_i \in R \\ 0 & \text{otherwise} \end{cases}$$

Recursive case: (Complete)

Return:

$$\pi(1, n, S) = \max_{t \in \mathcal{T}_G(s)} p(t)$$

Programming Problems (due March 12th)

Please read the submission instructions, policy and hints on the course webpage.

In this programming assignment, you will train a probabilistic context-free grammar (PCFG) for syntactic parsing. In class we defined a PCFG as a tuple

$$G = (N, \Sigma, S, R, q)$$

where N is the set of non-terminals, Σ is the vocabulary, S is a root symbol, R is a set of rules, and q is a probabilistic model. We will assume that the grammar is in Chomsky normal form (CNF). Recall from class that this means all rules in R will be either *binary rules* $X \rightarrow Y_1\ Y_2$ where $X, Y_1, Y_2 \in N$ or *unary rules* $X \rightarrow Z$ where $X \in N$ and $Z \in \Sigma$.

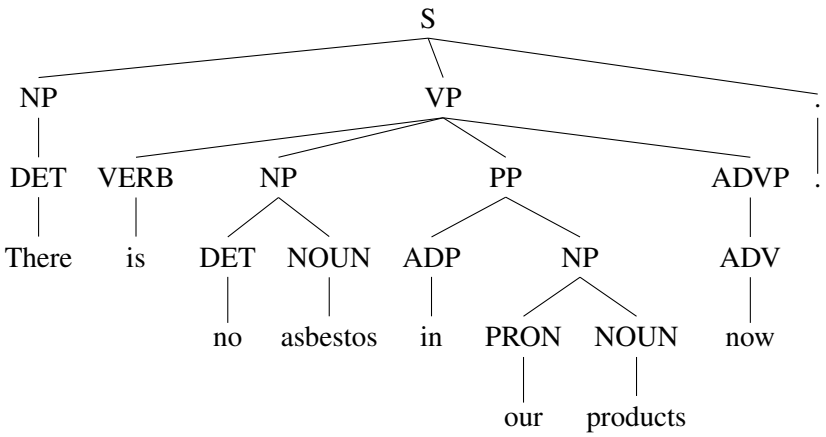
You will estimate the parameters of the grammar from a corpus of annotated sentences from the Wall Street Journal. Before diving into the details of the corpus format, we will discuss the rule structure of the grammar. You will not have to implement this section but it will be important to know when building your parser.

Rule Structure

Consider a typical tagged sentence from the Wall Street Journal

```
There/DET is/VERB no/DET asbestos/NOUN in/ADP our/PRON products/NOUN now/ADV ./.
```

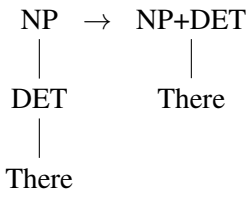
The correct parse tree for the sentence (as annotated by linguists) is as follows



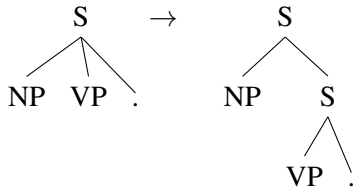
Note that nodes at the fringe of the tree are words, the nodes one level up are part-of-speech tags, and the internal nodes are syntactic tags. For the definition of the syntactic tags used in this corpus, see "The Penn Treebank: An Overview." The part-of-speech tags should be self-explanatory, for further reference see "A Universal Part-of-Speech Tagset." Both are linked to on the website.

Unfortunately the annotated parse tree is not in Chomsky normal form. For instance, the rule S $\rightarrow$ NP VP . has three children and the rule NP $\rightarrow$ DET has a single non-terminal child. We will need to work around this problem by applying a transfomation to the grammar.

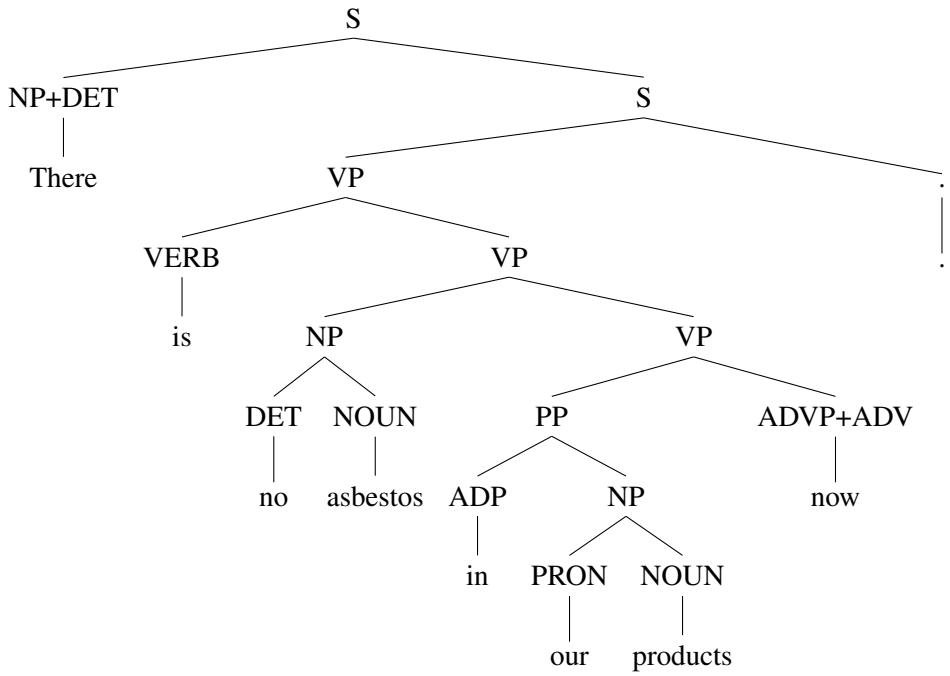
We will apply two transformations to the tree. The first is to collapse illegal unary rules of the form $X \rightarrow Y$ where $X, Y \in N$ into new non-terminals $X + Y$, for example



The second is to split n-ary rules $X \rightarrow Y_1\ Y_2\ Y_3$ where $X, Y_1, Y_2, Y_3 \in N$ into right branching binary rules of the form $X \rightarrow Y_1 X$ and $X \rightarrow Y_2 Y_3$ for example



With these transformation to the grammar, the new parse correct parse for this sentence is



Converting the grammar to CNF will greatly simplify the algorithm for decoding; however, as you saw in Question 1, there are other transformations into CNF that better preserve properties of the original grammar.

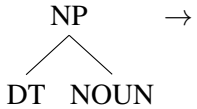
All the parses we provide for the assignment will be in the transformed format, you will not need to implement this transformation, but you should understand how it works..

Corpus Format

We provide a training data set with parses `parse_train.dat` and a development set of sentences `parse_dev.dat` along with the correct parses `parse_dev.key`. All the data comes for the Wall Street Journal corpus and consists of sentences of less than 15 words for efficiency.

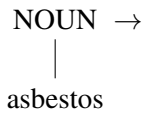
In the training set, each line of the file consists of a single parse tree in CNF. The trees are represented as nested multi-dimensional arrays conforming to the JSON standard. JSON is general data encoding standard (similar to XML). JSON is supported in pretty much every language (see http://www.json.org/).

Binary rules are represented as arrays of length 3.




```
["NP", ["DT", ...], ["NOUN", ...]]
```

Unary rules are represented as arrays of length 2.



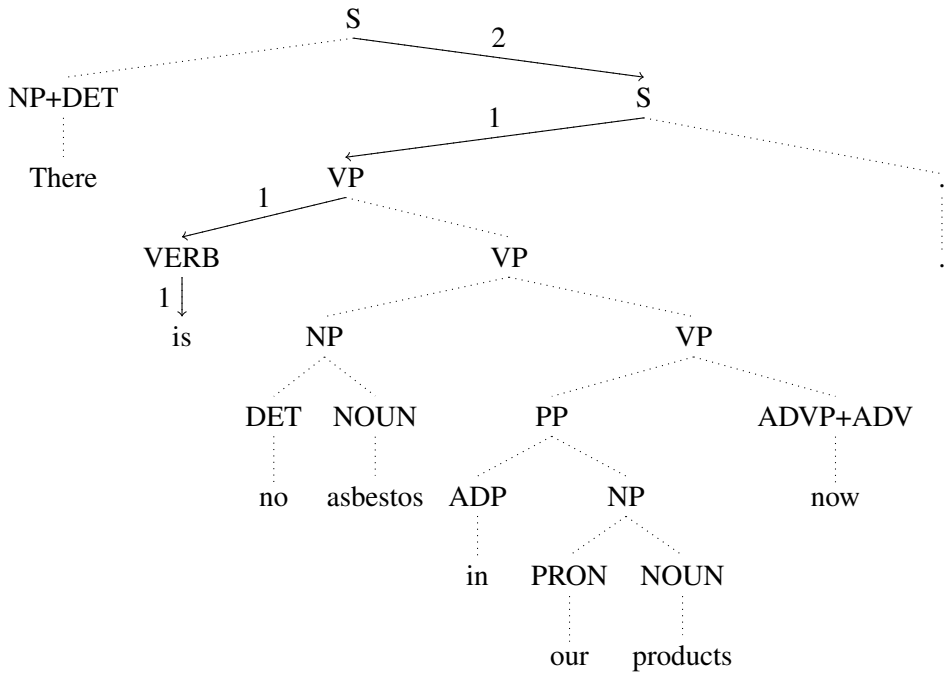
```
  ["NOUN", "asbestos"]
```

For example, the file `tree.example` has the tree given above

```
["S", ["NP", ["DET", "There"]], ["S", ["VP", ["VERB", "is"], ["VP", ["
   NP", ["DET", "no"], ["NOUN", "asbestos"]], ["VP", ["PP", ["ADP", "in
   "], ["NP", ["PRON", "our"], ["NOUN", "products"]]], ["ADVP", ["ADV",
    "now"]]]]], [".", "."]]]
```

The tree is represented as a recursive, multi-dimensional JSON array. If the array is length 3 it is a binary rule, if it is length 2 it is a unary rule as shown above.

As an example, assume we want to access the node for "is" by walking down the tree as in the following diagram.



In Python, we can read and access this node using the following code

```
import json
tree = json.loads(open("tree.example").readline())
print tree[2][1][1][1]
```

The first line reads the tree into an array, and the second line walks down the tree to the node for "is".

Other languages have similar (although possibly more verbose) libraries for reading in these arrays. Consult the TAs if you are unable to find a parser for your favorite language.

Tree Counts

In order to estimate the parameters of the model you will need the counts of the rules used in the corpus The script `count_cfg_freqs.py` reads in a training file and produces these counts. Run the script on the

training data and pipe the output into some file:

```
python count_cfg_freqs.py parse_train.dat > cfg.counts
```

Each line in the output contains the count for one event. There are three types of counts:

- Lines where the second token is `NONTERMINAL` contain counts of non-terminals $Count(X)$

  ```
  17 NONTERMINAL NP
  ```

  indicates that the non-terminal NP was used 17 times.

- Lines where the second token is `BINARYRULE` contain emission counts $Count(X \rightarrow Y_1\ Y_2)$, for example

  ```
  13 BINARYRULE NP DET NOUN
  ```

  indicates that binary rule NP $\rightarrow$ DET NOUN was used 13 times in the training data.

- Lines where the second token is `UNARYRULE` contain unigram counts $Count(X \rightarrow Y)$.

  ```
  14 UNARYRULE NP+NOUN products
  ```

  indicates that the rule NP+NOUN $\rightarrow$ products was seen 14 times in the training data.

Question 4 (20 points)

- As in the tagging model, we need to predict emission probabilities for words in the test data that do not occur in the training data. Recall our approach is to map infrequent words in the training data to a common class and to treat unseen words as members of this class. Replace infrequent words ($Count(x) < 5$) in the original training data file with a common symbol `_RARE_`. Note that this is more difficult than the previous assignment because you will need to read the training file, find the words at the fringe of the tree, and rewrite the tree out in the correct format. We provide a script `python pretty_print_tree.py parser_dev.key` which you can use to view your trees in a more readable format. The script will also throw an error if you output is not in the correct format. Re-run `count_cfg_freqs.py` to produce new counts.

  Submit your code and the new training file it produces.

Question 5 (40 points)

- Using the counts produced by `count_cfg_freqs.py`, write a function that computes rule parameters

$$q(X \rightarrow Y_1\ Y_2) = \frac{Count(X \rightarrow Y_1\ Y_2)}{Count(X)}$$

$$q(X \rightarrow w) = \frac{Count(X \rightarrow w)}{Count(X)}$$

- Using the maximum likelihood estimates for the rule parameters, implement the CKY algorithm to compute

$$\arg \max_{t \in \mathcal{T}_G(s)} p(t).$$

Your parser should read in the counts file (with rare words) and the file `parse_dev.dat` (which is just the sentence from `parse_dev.key` without the parse) and produce output in the following format: the tree for one sentence per line represented in the JSON tree format specified above. Each line should correspond to a parse for a sentence from `parse_dev.dat`. You can view your output using
`pretty_print_parse.py` and check your performance with `eval_parser.py` by running `python eval_parser.py parser_dev.key prediction_file`. The evaluator takes two files where each line is a matching parse in JSON format and outputs an evaluation score.

Note: There is an slight change from the class notes. Some of these sentences are fragments, and they do not have S as the root. For the purpose of this assignment you should change the last line of the CKY algorithm to

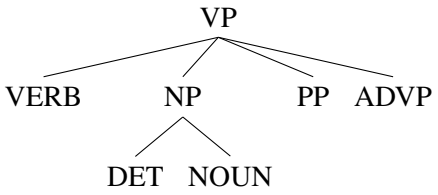
$$\text{if } \pi[1, n, S] \neq 0 \text{ return } \pi[1, n, S]$$
$$\text{else return } \max_{X \in N} \pi[1, n, X]$$

This should help performance.

Submit your program and report on the performance of your model. In addition, write up any observations you made.

Question 6 (20 Points)

Consider the following subtree of the original parse.



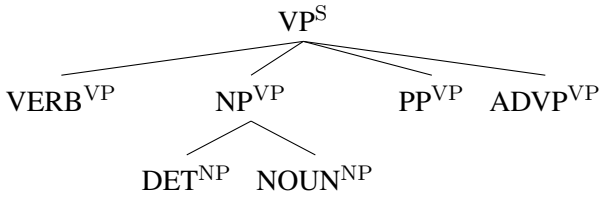
The tree structure makes the implicit independence assumption that the rule NP $\rightarrow$ DT NOUN is generated independently of the parent non-terminal VP, i.e. the probability of generating this subtree in context is

$$p(\text{VP} \rightarrow \text{VERB NP PP ADVP} \mid \text{VP}) p(\text{NP} \rightarrow \text{DET NOUN} \mid \text{NP})$$

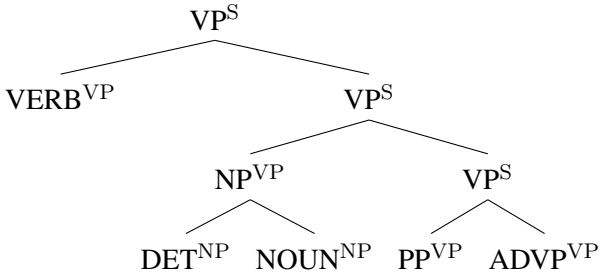
However, often this independence assumption is too strong. It can be informative for the lower rule to condition on the parent non-terminal as well, i.e. model generating this subtree as

$$p(\text{VP} \rightarrow \text{VERB NP PP ADVP} \mid \text{VP}) p(\text{NP} \rightarrow \text{DET NOUN} \mid \text{NP, parent=VP})$$

This is a similar idea to using a trigram language model, except with parent non-terminals instead of words. We can implement this model without changing our parsing algorithm by applying a tree transformation known as *vertical markovization*. We simply augment each non-terminal with the symbol of its parent.



We apply vertical markovization before binarization; the new non-terminals are augmented with their original parent. After applying both transformations the final tree will look like



There is a downside of vertical markovization. The transformation greatly increases the number of rules N in the grammar potentially causing data sparsity issues when estimating the probability model q. In practice though, using one level of vertical markovization (conditioning on the parent), has been shown to increase accuracy.

Note: This problem looks straightforward, but the difficulty is that we now have a much larger set of non-terminals N. It is likely that a simple implementation of CKY that worked for the last question will not scale to this problem. The challenge is to improve the basic inefficiencies of the implementation in order to handle the extra non-terminals.

- The file `parse_train_vert.dat` contains the original training sentence with vertical markovization applied to the parse trees. Train with the new file and reparse the development set. Run `python eval_parser.py parser_dev.key prediction_file` to evaluate performance.

  Report on the performance of your parser with the new trees. In what ways did vertical markovization help or hurt parser performance? Find an example where it improved your parser's output. Submit your program and any further observations.